Interspeech 2024
1-5 September 2024, Kos, Greece

Conformer without Convolutions

Matthijs Van keirsbilck[1], *Alexander Keller*[1]

[1]NVIDIA, Germany

matthijsv@nvidia.com, akeller@nvidia.com

Abstract

We analyze the weights of a trained speech-to-text neural network and discover a surprising amount of structure in the temporal convolutions. Based on our observations we propose to completely remove learnable temporal convolutions, and replace them with fixed averaging and shift operations which have no learnable parameters and open the way for significantly faster implementations. In the state-of-the-art models Conformer, Squeezeformer and FastConformer, this *improves* WER by 0.12%, 0.62%, and 0.20% respectively, while reducing the computational cost.

Index Terms: speech recognition, human-computer interaction, computational paralinguistics

1. Introduction

The fields of both speech recognition and computer vision produced a large body of feature extraction techniques, which are designed to be efficient, interpretable and are usually task-specific [1, 2, 3]. In the past decade, machine learning methods have replaced these handcrafted techniques by automatically learning to extract and interpret features. This can improve performance, but often at a significantly higher cost.

Previous work has examined weights learned by neural networks, with a particular focus on the first convolutional layer, and found that the learned filters resemble a traditional filter bank, specifically band-pass filters [4, 5]. Other work also analyzes the second layer [6], observing that many filters learn to detect differences and compute averages. However, exploiting this knowledge to improve training remains difficult, and training is still required.

Recently, fully-convolutional and attention-convolutional hybrid models have achieved impressive performance in both image and speech-to-text tasks. State-of-the-art computer vision [7] and speech recognition [8, 9, 10] networks rely on attention [11]. However, [12] and [13] found that averaging and shifting, respectively, can perform as well as the expensive attention mechanism for computer vision applications.

In this work we will analyze the trained weights of a simple fully-convolutional speech model, QuartzNet [14]. We will then demonstrate that it can be trained with no learnable convolutions across time but only fixed averaging and shift operations, which are simpler and computationally cheaper. We go on to show that our approach can be applied without any modification to state-of-the-art speech recognition models, improving word error rate (WER) and reducing computational cost.

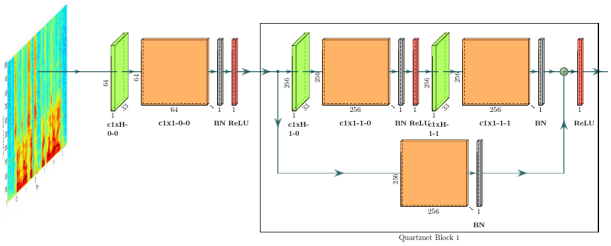

Figure 1: *The QuartzNet architecture up to the first residual block. Input to the network is a spectrogram computed using mel-frequency filter banks. The network consists a pre-encoder followed by residual blocks of convolutions across time (in green), linear transformations across features (in orange), BatchNorm and ReLU. A full block is indicated by the black rectangular outline.*

2. Preliminaries

QuartzNet [14] is a convolutional model that processes a spectrogram (mel-filterbank features) to produce a probability distribution over tokens from the vocabulary. The spectrogram has two dimensions: features and time. A visualization is shown in see Figure 1. A 'pre-encoder' convolution across time (in green) and a linear transformation across features (weight matrix plus bias, in orange) process the spectrogram. The output of these layers are passed to the first residual block. A block consists of a stack of R times a temporal convolution and linear transformation followed by BatchNorm [15] (BN) and ReLU [16]. For visualization, $R = 2$ Figure 1, while $R = 5$ in the actual network. In parallel with this stack there is a 'skip connection', i.e. another path with its own linear transformation, which is added to the output of the stack. The full architecture is a repetition of B of these blocks, reflected in the naming QuartzNet $B \times R$.

The temporal convolution consists of a set of 1D convolutional filters, one for each feature. The filters are convolved with the respective activations across the time dimension, hence the name 'temporal convolution'. The linear transformation with BN and ReLU can be seen as a single-layer neural network operating on the features, but not across time.

In this work, we will first investigate the relatively simple QuartzNet model. We will show that the learned convolutional filters are very simple operators, and that those can be learned by the linear transformation that follows each of the temporal convolutions. Second, we propose to remove the learnable temporal convolutions entirely, and provide the linear transformation with only averages and shifted copies of the input (requiring no learning at all). Third, we show that the same idea can be applied to state-of-the-art speech recognition models, simplifying the architecture and improving WER even though we remove learnable layers entirely.

10.21437/Interspeech.2024-445

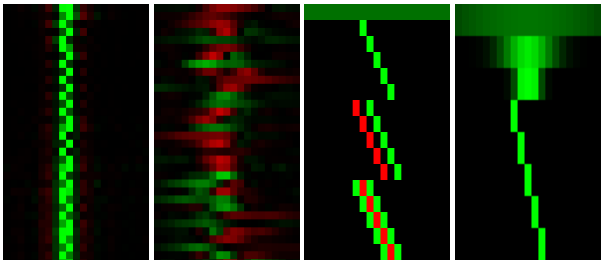

Figure 2: *Each row visualizes the weights of a 1D temporal convolutional filter, which will be applied to one feature dimension across time. Different rows correspond to different feature dimensions. Negative weights are red, positive weights are green, and black is zero.* ***Left and middle left****: filter weights in trained QuartzNet, the first layer in pre-encoder and in 4th residual block.* ***Middle Right:*** *filters implementing (top to bottom) averaging, shifted copies, and shifted 1^{st} and 2^{nd} derivatives, with 5 shifts $\{-2, -1, 0, +1, +2\}$. Shift 0 corresponds to copying the input.* ***Right:*** *filters implementing (top to bottom) Gaussian averaging of full, half and quarter width, and shifted copies.*

We use the NeMo toolkit [17] to train our models on Librispeech (configuration files available online). For cheaper experimentation we use as a baseline QuartzNet10x5-5, a cutdown version of QuarztNet15x5 [14] where we replace the last five blocks with one block of 5 1×1 convolutions layers with BN and ReLU, but no temporal convolutions. This model achieves quite similar WER at much lower cost compared to the 15x5 model: WER is 4.28% (4.04%), parameter count is 11.4×10^6 (18.9×10^6) and training time 35.7 hours (52.7 hours) as shown in Table 4. Authors investigating Transformer models [18] have shown that attention matrices in the last layers are close to identity, indicating that operations across time are not important in these final layers, which may explain why this simplified model performs quite well.

3. Removing Temporal Convolutions

3.1. Analyzing Trained Temporal Convolutions

The trained weights of the temporal convolutions of QuartzNet exhibit clear structure. Some look like averaging and copy operators, or derivatives. Others resemble Gabor functions [19].Observations in neuroscience confirm the existence of similar operations in the visual cortex [20, 21].

As an example, weights of the first temporal convolution and in the 4^{th} residual block are visualized in the left and center left images in Figure 2. In almost all rows weights at the borders are close to zero and the largest magnitudes can be found near the center of the filters. Many of the filters compute a sort of average of the input, sometimes with negative sign. Other rows have positive weights on the left and negative on the right, meaning the filter computes edges or differences. These observations closely match the analysis of [6]. Lastly, many filters are not centered. We will revisit this in Section 3.3.2.

Each temporal convolution is followed immediately by a linear transformation (see Figure 1), which allows for linear equivalence transformations. For example, we can multiply all weights in a row of the temporal convolution if the weights in the corresponding column of the following weight matrix are multiplied with the inverse of the scalar. In this manner, we can demonstrate that many filters differ from each other only by a scale, indicating significant redundancy in these learned temporal convolutions. As we will show, this redundancy will allow us to remove the learned temporal convolutions entirely.

Figure 3: *Visualizing the first 12 rows in Figure 2 (left), i.e. 1D temporal convolutional filters of width 33 in the first temporal convolution in QuartzNet after training. The Y-axis range is [-0.3, 1.1] for all plots. All filters are very similar.*

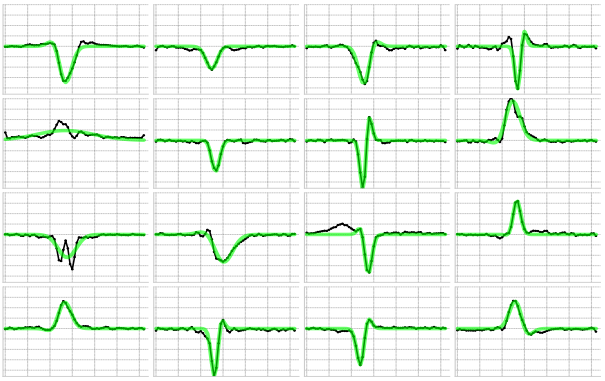

Figure 4: *Visualizing the first 16 rows in Figure 2 (middle left), i.e 1D temporal convolutional filters of width 63 in the 4^{th} QuartzNet block after training. Each graph shows the trained filter weights (black), and a Gabor function fitted to the filter weights using the least-squares method (green). The Y-axis range is [-0.23, 0.20] for all plots. Gabor functions can approximate the trained filters well.*

Looking at the very first temporal convolution in QuartzNet, we can multiply some rows with -1, flipping the sign. Then, we divide each row by its maximum value to be able to compare the learned filters. The result is visualized in leftmost image of Figure 2 as well as in Figure 3. We see clearly that all 1D convolutions compute approximately the same function except for a shift, resembling a wavelet or sharpening filter. We will show that learned filters in all layers can be well approximated with simple Gabor functions.

3.2. Fitting Gabor Functions

Gabor functions [19] are the product of a cosine/sine pair and a Gaussian function, parametrized by the center frequency f and variance σ^2 (showing only the cosine component):

$$g(x; f, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}} \cos(2\pi f x) \quad (1)$$

To confirm our intuition that the learned filters match Gabor functions, we attempt to fit these parameters to the learned weights using the least-squares method. Figure 4 shows that this works well for most of the filters.

Based on these results one could learn Gabor parameters directly similar to [22], though the optimization landscape is difficult [23]. One might compress the weights by storing only the Gabor parameters instead of all individual filter weights. We will however take a different approach, as described below.

Table 1: *QuartzNet10x5-5 (20 epochs), overview of models with learned temporal convolutions or fixed averages and shifts (37.5% averages of 3 different widths, 62.5% copies with 5 shifts)*

	WER (%)
fully trainable temporal convolutions	5.71
fixed shifts, averages in blocks 1-10	6.72
fixed shifts, averages in blocks 1-8	**5.65**

Table 2: *QuartzNet10x5-5 (20 epochs) comparing simple operations replacing the temporal convolutions.* ***Left:*** *averaging, (shifted) copies, 1^{st} and 2^{nd} derivatives with 0 and 5 shifts.* ***Right:*** *only shifted copies, varying the number of shifts. Adding shifts dramatically improves WER.*

	WER (%)	
	no shifts	5 shifts
only averages	22.10	N/A
only copies	26.90	8.05
only 1^{st} derivatives	18.25	16.41
only 2^{nd} derivatives	23.14	20.61
75% averages, 25% copies		9.37
50% averages, 50% copies		8.33
25% averages, 75% copies		**7.55**

	WER (%)
no shifts	26.90
3 shifts	9.10
5 shifts	**8.05**
7 shifts	8.40
9 shifts	8.53

3.3. Replacing convolutions with simple operations

Starting from our observations, we will exploit the fact that each temporal convolution is followed by a 1×1 convolution plus activation, i.e. a one-layer neural network which can compute any function of the features[24], including permutations. We can replace the temporal convolution with averaging and shifting operations, and since permutations can be learned, it does not matter to which features the averaging and shifting is applied, i.e the network can learn to assign operations to features. Only the very first convolution layer in the pre-encode part of the network (see Figure 1) is not preceded by a 1×1 convolution, so we don't modify this layer. For these investigations we report dev-clean WER after 20 training epochs on Quartznet10x5-5, training with 8 NVIDIA V100 GPUs.

3.3.1. Copies, Averaging, and Derivatives

The previous section makes apparent that many of the learned 1D filters resemble simple linear operations like averaging, copies, and derivatives. Hence, we investigate how well the model performs if we use exactly those operations instead of learnable temporal convolutions. A visualization of these operations as 1D convolutions with specific weights is shown in Figure 2 (center right). An averaging filter of width W timesteps uses $1/W$ for all weights, while a filter implementing a copy has all zero weights except for a 1.0 in the center. To compute derivatives we use central finite differences as they provide second order accuracy [25]. The left part of Table 2 shows that all variants perform badly with no shifts.

3.3.2. Shifts across timesteps

Figure 2 shows that most learned filters are not centered, so they shift their input left or right in the time dimension. The next layer then has access to features from multiple timesteps, which may be very useful for sequential data like speech. We propose to add shifts to the fixed functions. Using shifts to mix features across time resembles TSM [26] where shifts are to mix information across video frames, and ShiftNet [27] where shifts replace spatial convolutions. We implement $2N + 1$ shifts in 1D convolutions by offsetting the weights in the respective filters from the center by $\{-N, \ldots, 0, \ldots, +N\}$ steps. Figure 2 (middle right) visualizes averaging and shifted copies, 1^{st} and 2^{nd} derivatives with 5 shifts. Table 2 shows that adding even a few shifts improves the WER dramatically.

The layer following the shifted copies has access to features across multiple timesteps, so it can learn to compute differences and providing 1^{st} and 2^{nd} derivatives becomes redundant. Removing those improves WER, leaving us with only averages and shifts. Table 2 shows that the WER is lowest with 25% averages and 75% shifts.

Table 3: *QuartzNet10x5-5 with copies using 5 shifts and averages of varying width W in all encoder blocks*

	WER (%)
width $0.1 \times W$	8.02
width $0.25 \times W$	7.75
width $0.5 \times W$	7.48
default width ($1.0 \times W$)	7.55
width $2.0 \times W$	7.18
width $4.0 \times W$	6.97
width $8.0 \times W$	7.44
width $W, W/2, W/4$	6.95
Gaussians, width $W, W/2, W/4$	**6.72**

3.3.3. Averaging from a Filtering Perspective

We now take a closer look at the averaging operations. We first discuss the impact of varying the width of the averaging operator, then show how combining averages of different widths improves WER, and finally show that using Gaussian averaging further improves the model.

First, an averaging convolution acts like a low-pass filter, where wider averages show stronger low-pass behavior. In the case of speech spectrograms, a low-pass filter across time may give the network an estimate of the average power per feature, which it can use to compare with the power in the current time step. For the default QuartzNet filters are between 33 and 75 timesteps wide, increasing from the first to the last blocks in the the network. Increasing the width of an averaging filter is essentially free (see Section 3.4.1), in contrast to increasing the width of learned convolutions. For example, using 8x wider temporal convolutions increases the training time from 4.6h to 12.4h. Table 3 shows that wider averages of around $4\times$ the default give the best WER (corresponding to filters of 132 to 250 timesteps wide).

Second, [6] showed that trained convolutional filters have significantly different frequency response, leading us to use averages of three different widths: $W, W/2, W/4$ where W is the width of the original convolutional filter. With three different averages and five different shifts $(-2, -1, 0, +1, +2)$ we now have a total of eight different operations, each applied to $\frac{1}{8}$ of features. A visualization is shown in the rightmost part of Figure 2. Table 3 shows that this configuration reaches a WER of 6.95%

Third, we show that Gaussian averaging further improve the WER. Rectangular averaging filters in the time domain correspond to sinc functions in the Fourier domain, which may introduce artifacts limiting performance of the network, and can be mitigated by using Gaussian averaging filters instead. We

select their width σ so that 4σ equals the width of the corresponding rectangular average. WER improves to 6.72%, or only 1% worse than the fully trained network. We note that Gaussians functions are closely related to the Gabor functions we discussed in Section 3.2, and using Gaussians of different widths is reminiscent of classical Gaussians pyramids [28] as well as filter banks [6].

This is our final configuration and what we use in all experiments of Section 4. In contrast to classical filter banks which generally must be redesigned for each model and dataset, we show below that we can use the exact same configuration of shifted copies and averages for different models.

3.3.4. Learning some Temporal Convolutions

We note that QuartzNet with averages and shifts instead of learned convolutions does not match the original WER. However, by allowing learning in a few layers we can significantly reduce the gap with the fully-trained model while still benefiting from the efficiency gains. We found it especially important to learn the convolutions at the end of the network, in the last two blocks. Table 1 shows that this hybrid variant improves the WER from 6.72% to 5.65%, which is *lower* than the 5.71% for network with learned temporal convolutions.

3.4. Discussion

We found that trained filters can be well categorized as with simple operations: shifts, averages, edge detectors, and as a generalized function class, Gabor functions. We then showed QuartzNet with only extremely simple shifts and averages can reach a WER close to the network with fully trained temporal convolutions. We use the same operations in all network blocks, replacing over 50 convolutional layers. While these layers contain only few parameters compared to the 1×1 convolutions and linear layers, it is remarkable that there is no degradation in WER.

The results in Section 3.3.2 give an indication why this is possible: the shift operations are absolutely essential. Each of the time operations is followed by single-layer neural network (linear transformation, BatchNorm, ReLU). The shifts provide this layer with features from multiple timesteps, allowing it to compute functions across the temporal dimension [24, 26]. In a sense we show that this layer can then perform the task of the temporal convolution, which explains why the convolution can be removed.

3.4.1. Note on Efficient Implementation

Our implementation does not modify the model at all, and implements averages and shifts by setting weights in the 1D convolutions. This is inefficient as the convolutions copy the activations while shifts only require using a pointer offset in memory [27], which would avoid the unnecessary (and expensive) memory traffic. The averages can also be computed much more efficiently than using a convolution, e.g. via a moving average.

Moreover, since our layer does not require any training, it is unnecessary to store the activations or gradients for backpropagation and the layer may be fused with the following Linear layer. This can result in $2\times$ memory savings for activations and significant speedups for fully-convolutional models like QuartzNet, since around half of the layers are temporal convolutions.

Table 4: *Averaging and shifts replacing learned convolutions*

	WER test (%)		Weights	Training
	clean	other	$\times 10^6$	hours
QuartzNet 15x5	4.04	11.44	18.9	52.7
QuartzNet 10x5-5	4.28	12.30	11.4	35.7
", fixed shifts/averages	4.86	14.28	10.8	30.6
", learn temp. conv in blocks 9-10	4.45	12.95	11.1	33.0
ConformerS RNN-T	3.54	8.71	14.0	137.4
", fixed shifts/averages	3.51	8.59	14.0	132.2
SqueezeformerXS RNN-T	3.58	8.63	10.1	98.5
", fixed shifts/averages	3.29	8.01	10.0	97.2
FastConformer-L CTC	2.79	6.72	108.0	85.2
", fixed shifts/averages	2.77	6.52	107.9	84.0

4. Experimental results

Table 4 shows the results for full training on Librispeech for QuartzNet and for the newer models Conformer [8], Squeezeformer [9] and state-of-the-art FastConformer [10] which combine attention with convolutional and feedforward layers. We modify these in exactly the same way as QuartzNet: we remove all learned temporal convolutions and replace them with three different averages and five different shifts.

We use standard training configurations and hyperparameters as provided in NeMo[1] following [14], using SentencePiece [29] unigram tokenization with 128 tokens and training QuartzNet, Conformer and Squeezeformer for 400 epochs and FastConformer for 200 epochs.

QuartzNet10x5-5 with fixed shifts and averages in all 10 blocks has a WER of 4.86%, compared to 4.28% for the fully-trained model. The WER improves to 4.45% when using learnable temporal convolutions in the last two blocks.

For the more recent attention-based models, WER *improves* when removing learnable temporal convolutions: test-clean WER and test-other WER improve in Conformer by 0.03% and 0.12% , in Squeezeformer by 0.29% and 0.62% , and in FastConformer by 0.02% and 0.20% . These results show that our approach transfers well to non-convolutional models, and in fact even improves them.

We note again that the cost of running a model can be reduced significantly especially for fully-convolutional models like QuartzNet, which may be used in low-power applications. This is already visible in the training times in Table 4, and significant further reductions in cost are possible as discussed in Section 3.4.1.

5. Conclusion and Future Work

We demonstrate existing redundancy in popular models for speech recognition and show that all trainable temporal convolutional layers in state-of-the-art speech recognition models can be replaced by extremely simple and computationally cheap fixed averaging and shift operations. This decreases their computational cost to run these models and improves their accuracy at the same time.

Combining this with other work on more efficient attention mechanisms [10, 30, 31] and more efficient alternatives to fully-connected feedforward layers [32, 33] may lead to much cheaper and more performant speech recognition models.

[1] https://github.com/NVIDIA/NeMo/tree/main/examples/asr/conf

6. References

[1] N. Dave, "Feature extraction methods LPC, PLP and MFCC in speech recognition," *International Journal for Advance Research in Engineering and Technology*, 2013.

[2] M. Benzeghiba, R. De Mori, O. Deroo, S. Dupont, T. Erbes, D. Jouvet, L. Fissore, P. Laface, A. Mertins, C. Ris *et al.*, "Automatic speech recognition and speech variability: A review," *Speech communication*, 2007.

[3] H.-B. Deng, L.-W. Jin, L.-X. Zhen, J.-C. Huang *et al.*, "A new facial expression recognition method based on local Gabor filter bank and PCA plus LDA," *International Journal of Information Technology*, 2005.

[4] Z. Tüske, P. Golik, R. Schlüter, and H. Ney, "Acoustic modeling with deep neural networks using raw time signal for LVCSR," in *Fifteenth annual conference of the international speech communication association*, 2014.

[5] D. Palaz, R. Collobert *et al.*, "Estimating phoneme class conditional probabilities from raw speech signal using convolutional neural networks," in *Proceedings of Interspeech*, 2013.

[6] P. Golik, Z. Tüske, R. Schlüter, and H. Ney, "Convolutional neural networks for acoustic modeling of raw time signal in LVCSR," in *Sixteenth annual conference of the international speech communication association*, 2015.

[7] S. Khan, M. Naseer, M. Hayat, S. W. Zamir, F. S. Khan, and M. Shah, "Transformers in vision: A survey," *ACM computing surveys (CSUR)*, 2022.

[8] A. Gulati, J. Qin, C.-C. Chiu, N. Parmar, Y. Zhang, J. Yu, W. Han, S. Wang, Z. Zhang, Y. Wu *et al.*, "Conformer: Convolution-augmented transformer for speech recognition," *Interspeech*, 2020.

[9] S. Kim, A. Gholami, A. Shaw, N. Lee, K. Mangalam, J. Malik, M. W. Mahoney, and K. Keutzer, "Squeezeformer: An efficient transformer for automatic speech recognition," *Advances in Neural Information Processing Systems*, 2022.

[10] D. Rekesh, N. R. Koluguri, S. Kriman, S. Majumdar, V. Noroozi, H. Huang, O. Hrinchuk, K. Puvvada, A. Kumar, J. Balam *et al.*, "Fast conformer with linearly scalable attention for efficient speech recognition," in *IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, 2023.

[11] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in Neural Information Processing Systems*, 2017.

[12] W. Yu, M. Luo, P. Zhou, C. Si, Y. Zhou, X. Wang, J. Feng, and S. Yan, "Metaformer is actually what you need for vision," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022.

[13] G. Wang, Y. Zhao, C. Tang, C. Luo, and W. Zeng, "When shift operation meets vision transformer: An extremely simple alternative to attention mechanism," *Proceedings of the AAAI Conference on Artificial Intelligence*, 2022.

[14] S. Kriman, S. Beliaev, B. Ginsburg, J. Huang, O. Kuchaiev, V. Lavrukhin, R. Leary, J. Li, and Y. Zhang, "QuartzNet: Deep automatic speech recognition with 1d time-channel separable convolutions," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2020.

[15] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*, 2015.

[16] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *European Conference on Computer Vision (ECCV)*, 2016.

[17] O. Kuchaiev, J. Li, H. Nguyen, O. Hrinchuk, R. Leary, B. Ginsburg, S. Kriman, S. Beliaev, V. Lavrukhin, J. Cook *et al.*, "NeMo: a toolkit for building AI applications using neural modules," 2019. [Online]. Available: https://github.com/NVIDIA/NeMo

[18] S. Zhang, E. Loweimi, P. Bell, and S. Renals, "On the usefulness of self-attention for automatic speech recognition with transformers," in *IEEE Spoken Language Technology Workshop (SLT)*, 2021.

[19] D. Gabor, "Theory of communication. Part 1: The analysis of information," *Journal of the Institution of Electrical Engineers-Part III: Radio and Communication Engineering*, 1946.

[20] J. G. Daugman, "Two-dimensional spectral analysis of cortical receptive field profiles," *Vision research*, 1980.

[21] J. P. Jones and L. A. Palmer, "An evaluation of the two-dimensional Gabor filter model of simple receptive fields in cat striate cortex," *Journal of Neurophysiology*, 1987.

[22] N. Zeghidour, O. Teboul, F. d. C. Quitry, and M. Tagliasacchi, "LEAF: A learnable frontend for audio classification," *International Conference on Learning Representations*, 2021.

[23] B. Hayes, C. Saitis, and G. Fazekas, "Sinusoidal frequency estimation by gradient descent," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2023.

[24] B. Hanin, "Universal function approximation by deep neural nets with bounded width and ReLU activations," *Mathematics*, vol. 7, no. 10, p. 992, 2019.

[25] B. Fornberg, "Generation of finite difference formulas on arbitrarily spaced grids," *Mathematics of Computation*, 1988.

[26] J. Lin, C. Gan, and S. Han, "TSM: Temporal shift module for efficient video understanding," in *CVF International Conference on Computer Vision*, 2019.

[27] B. Wu, A. Wan, X. Yue, P. Jin, S. Zhao, N. Golmant, A. Gholaminejad, J. Gonzalez, and K. Keutzer, "Shift: A zero flop, zero parameter alternative to spatial convolutions," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[28] P. J. Burt and E. H. Adelson, "The Laplacian pyramid as a compact image code," *IEEE Transactions on Communications*, vol. 31, pp. 532–540, 1983.

[29] T. Kudo and J. Richardson, "SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing," *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (System Demonstrations)*, 2018.

[30] A. Gupta, A. Gu, and J. Berant, "Diagonal state spaces are as effective as structured state spaces," *Advances in Neural Information Processing Systems*, 2022.

[31] H. Liu, M. Zaharia, and P. Abbeel, "Ring attention with blockwise transformers for near-infinite context," in *NeurIPS Foundation Models for Decision Making Workshop*, 2023.

[32] S. Jaszczur, A. Chowdhery, A. Mohiuddin, L. Kaiser, W. Gajewski, H. Michalewski, and J. Kanerva, "Sparse is enough in scaling transformers," *Advances in Neural Information Processing Systems*, 2021.

[33] T. Gale, D. Narayanan, C. Young, and M. Zaharia, "Megablocks: Efficient sparse training with mixture-of-experts," *Proceedings of Machine Learning and Systems*, 2023.